Chapter 8

Risk Management

Without risk, there is no reward.

Most software projects are generally one-off projects; that is, the problem being solved has not been solved before. Each project requires different resources and has different constraints and objectives. As a result, one of the few things we can be certain about in most software projects is that there will be uncertainty.

Rather than crossing our fingers and hoping that nothing bad happens, the best way to deal with uncertainty is to plan for it and manage it. We do this using *risk management*. A project management team that understands the nature of the risks involved in a project and is equipped to manage them can work with less resources and less contingency than a team that is less prepared.

8.1 Risk, Uncertainty and Risk Exposure

Risk is part of our everyday lives. Simply leaving the house in the morning is a risk — who knows what may happen on our way to university each day? We deal with these risks using a variety of techniques, but for the most part, these techniques are unstructured and informal — although they suffice.

The term "risk" is also part of our everyday language. However, for the purpose of formal risk management, we give it a precise definition.

Definition 8.1. Risk

A common definition of "risk" is as follows:

A risk is a possible future event that has negative results.

This implies that the result of the event is always harmful (a *downside risk*), however, we can just as easily include events in which the result is beneficial (an *upside risk*).

The above definition is generally used because it seems strange to deal with a positive event as a risk. However, if we plan for possible future events that have positive results, then we can take advantage of them. A trivial example involves finishing an activity ahead of time. If we are not equipped to start dependent activities when an activity finishes, then we will lose some benefit of finishing early.

Thus, we redefine risk as follows:

A risk is a possible future event that has some expected impact.

In this definition, the expected impact can be either negative or positive; harmful or beneficial.

It may still seem strange to consider events with positive impacts as risks. One may instead consider the risk of such an event being the risk that we *fail to take advantage of the upside*.

One must note that risk is different to *uncertainty*. Uncertainty is, as its name implies, the lack of complete certainty. As such, we say that we are uncertain about an event if we assign a probability of less than 1 of that event occurring. However, uncertainty is not concerned with the expected impact of an event. For example, we can be uncertain about the outcome of a sporting event, however, unless we have placed a bet on that event, or have some other personal stake in it, there is no risk. Therefore, we can have uncertainty without risk, but not risk without uncertainty. A risk with a probability of 1 is a *problem*.

There are many events that happen during the course of a project. We can determine which events are risks by analysing the following three properties of the risk:

1. the *probability* of the event occurring;
2. the *impact* that the event would have on the project; and
3. the *degree of control* we have over the event or its outcome.

If an event has a possibility of occurring, will have an impact on our project if it does occur, and there is something we can do about this, then this event is a risk for our project.

Definition 8.2. Risk exposure

The *risk exposure* of an event is defined as follows:

$$\textit{Risk exposure} \;=\; \textit{probability} \times \textit{impact}.$$

That is, the exposure of a project to an event is defined as the probability of that event occurring, multiplied by the impact that event will have on the project. The probability is a measure between 0 and 1 inclusive.

The impact can be measured using different scales, such a finite grade of 1-5, in which the grades mean: (1) no impact; (2) minimal impact; (3) moderate impact; (4) severe impact; and (5) catastrophic impact; or could be measured using monetary cost or time cost. What is important is that the impact is *quantifiable*, so that the risk exposure can be calculated. For example, a risk with probability 0.2 with an impact of "yeah, pretty bad" has no risk exposure, and can not be determined to be a higher or lower priority that a risk with probably 0.15 with an impact of "oooo... you do not want that to happen". However, if we rate the impact on a quantifiable scale, such that the first risk has an impact of 2 (minimal) while the second has an impact of 5, then the risk exposures are 0.4 and 0.75 respectively. From this, we determine that the second risk is greater.

A *risk-free* event is one in which the risk exposure is zero; that is, either the probability or the impact are zero.

Example 8.1. *Third-party software applications.*

As an example, consider the example of using a third-party software application to provide some functionality of a system that is being developed. There are several risks association with this, such as the following:

1. If the third-party application is being developed in parallel with the system under development, there is a risk that it could be delivered later than planned, thereby delaying the delivery of the entire system.
2. Once complete, the third-party application may not be reliable enough to be used, meaning that a new third-party application providing the functionality will need to be sourced or developed.
3. The third-party application may deliver behaviour that is inconsistent with the the expectations of the system developers, meaning that a new third-party application providing the functionality will need to be sourced or developed.

To plan for and manage these risks, we must estimate the probability and the impact. The table below shows these estimates, with impact measured a monetary cost, along with the risk exposure to each event.

Event	**Prob.**	$\times$	**Impact**	=	**Risk Exposure**
Late delivery	0.15		\$10,000		\$1500
Unreliable	0.05		\$20,000		\$1000
Inconsistent	0.2		\$20,000		\$4000

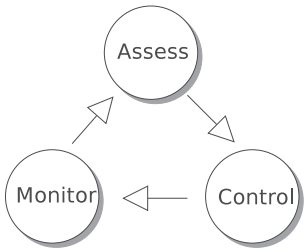

Figure 8.1: The iterative process of risk management.

From the risk exposure estimates, we would say that the third-party application behaving inconsistently with our expectations is the largest risk of the three, followed by late delivery, and finally, the third-party application being too unreliable.

Example 8.2. *Great risk management failures.*

The following is a good example of a failure to properly manage risk in a non-software domain. From the period 2000-2007, the market for credit default swaps — basically insurance against default of loan and bond payments — grew from US$631 billion to US$46 *trillion*. Many of the main players in this market were offering this insurance against collections of U.S. residential mortgages. However, these players made two fatal mistakes. First, they underestimated the probability of non-trivial amounts of defaults, with some of them ignoring the chance of a housing market collapse completely (thus claiming that investing in residential mortgages or residential property was almost risk free, despite the high returns it offered). Second, many of those that did assign a non-zero probability to loan defaults then ignored the risk overall because their probability estimate was so small. This was a mistake because the impact was devastating, and likely to send them bankrupt. That is, they used risk probability instead of risk exposure to prioritise some of their risks.

American International Group (AIG) were a major player in this market, and as a result of their failings to properly assess risk, they filed for bankruptcy protection in September 2008, and were only able to continue operating thanks to a "re-capitalisation" of US$85 billion from the U.S. government. Clearly, in this case, the risk managers failed to assess the risks correctly[1].

8.2 Risk Management Activities

Definition 8.3. Risk Management

To reduce the chances of an event causing problems on a project, we use *risk management*. Risk management is a structured process for dealing with risk within a project, as opposed to an ad-hoc approach. There are many viewpoints on how risk management should be properly performed, however, most are broken into the following two broad categories:

1. *Risk assessment:* identifying and analysing the risks to the project, and prioritising these risks based on our exposure to them.

2. *Risk control:* removing, mitigating, accepting, or minimising the risks to the project.

Risk management is not a task that is undertaken only at the initial stages of a project. It is an iterative and ongoing activity, in which one is continually assessing, controlling, and monitoring the risks in the project, as demonstrated by Figure 8.1. Note the additional activity of monitoring the project so that, if a risk arises, we can identify this and enact the plan for dealing with it. This typically consists of the activities discussed in Part III of these notes, so they get no further discussion in this chapter.

Figure 8.2 outlines the activities that further break up risk assessment and risk control according to Rook [Roo93].

[1] Although, considering the U.S. government covered AIG's losses, one could say that their assumption of a low risk exposure was valid! This is a perfect example of *moral hazard*.

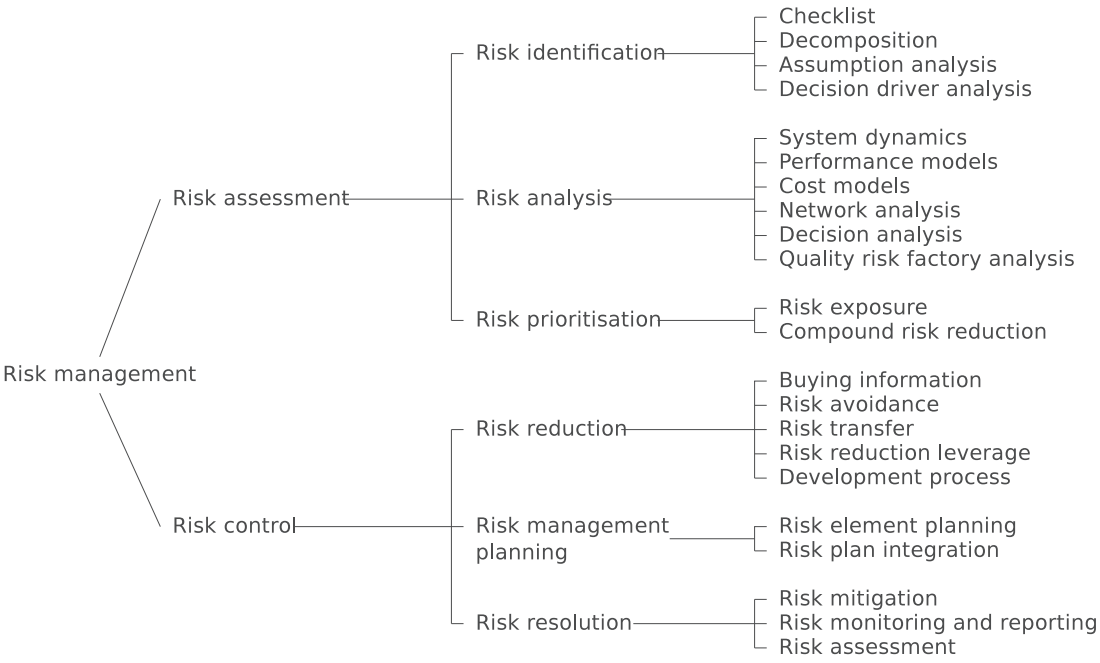

Figure 8.2: The hierarchy of activities involved in risk management according to Rook [Roo93].

Remark. It can be argued that all project management is actually risk management. For example, consider software testing. If we do not test a piece of software before it is released, there is a (very high) risk that the faults in the software will cause our users to be dissatisfied. To mitigate this risk, we systematically test the system before we release it to find as many faults as possible, and then remove them.

Decades of experience in software development has demonstrated that this risk occurs in *every* project, so it is implicitly mitigated by software development life cycle models. Such risks are known as *generic risks* In fact, every process in a software engineering project mitigates (or avoids) a risk: we engineer requirements systematically because our experience tells us that starting with programming results in us building the wrong systems; we use source-code repositories because our experience tells us that only keeping local copies causes many headaches.

On the other hand, there are many risks that are relevant only to a particular project; for example, using a particular third-party application from a particular vendor. These are known as *specific risks*. The job of a risk manager is to assess and control specific risks. The methods of controlling specific risks are often software engineering process themselves. For example, if there is a risk that a third-party application will not work as expected, we may design the system such that a generic interface is placed between our system and the third-party system, so that if the third-party application changes, the change to our system is minimised.

8.2.1 Risk Assessment

The first broad category of risk management is risk assessment. From Figure 8.2, we can see that risk assessment consists of *identifying* the risks, *analysing* the risks, and *prioritising* the risks.

Risk Identification

Risk identification is somewhat of a creative activity that requires experience, lateral thinking, and a good understanding of the project. We need to identify the cause of the risky event, the consequence of the event, the probability of the event occurring, and the impact that the event would have on the project. Some risk assessment processes also specify that we should recommend a preliminary course of action should the event occur.

There are many different techniques for identifying risks on projects. It is likely that none of these is sufficient in isolation for a single project. Instead, a combination of several techniques would be the most effective way to identify real risks. Some techniques for risk identification include the following:

- **Pondering** — this simply involves an individual taking the "pencil and paper" approach of risk identification, which involves sitting and thinking about the possible risks that could occur in the project. This is one of the initial risk assessment tasks used in many projects.
- **Interviews/questionnaires** — this involves interviewing experts, or asking them to fill out questionnaires, to harness their knowledge of a domain. It is unlikely that a risk manager in a software project will have sufficient knowledge of the methods and tools to be employed to provide a comprehensive view of the risks, so input from domain experts is essential.
- **Brainstorming** — this is a an activity in which a group of experts (generally six to twelve) from different backgrounds generate a large number of possible risks. Crazy ideas are encouraged early in brainstorming sessions, on the basis that ideas are easier to modify than originate, and that someone from a different background may link the crazy idea to a new idea that is less crazy.
- **Checklists** — this involves the use of standard checklists of possible risk drivers that are collated from experience. These checklists are used a triggers for experts to think about the possible types of risks in that area. Checklists can be general (e.g. the risks associated with cost estimates), or specific (e.g. the risks associated with using a particular software library).

Chapman and Ward [CW96] advocate trying to make risk assessment fun, and to treat risks as "good things". Not only does this mean that people doing risk assessment are less likely to become bored; it will also improve the risk management overall. Chapman and Ward argue that, if risks are treated as inherently bad, those people performing risk assessment will take a blinkered view of the domain, especially if they have some stake in the project other than as a risk assessor.

Human beings excel in discounting and ignoring real threats to themselves, especially if the impact of those threats are large. Consider the case of the credit default swaps discussed in Example 8.2 — risk assessors in these financial companies ignored risks because the impact was too devastating for them to consider; they would lose their multi-million dollar bonuses if something went wrong!

If risks are seen as a being good, and finding them is seen as positive, then people will remove their "blinkers" and look for opportunities to find risks.

Such techniques require creative and lateral thinking to be successful, so Chapman and Ward warn against the over-reliance on checklists and questionnaires. They also point out that good morale in a project is important to performing good risk management, as this reduces down cultural and communication problems between sub-teams within the project.

Example 8.3. *Hazard and operability studies (HAZOP).*

A HAZOP study [Kle86] is a structured process for the identification of hazards and risks contained in a process or operation. Originally developed to identify hazards in chemical process systems, it has been used in software engineering as a hazard identification technique in safety-critical systems. HAZOP is a combination of keyword-driven risk assessment, brainstorming, and pondering.

Given a process, a HAZOP study proceeds by applying a set of *guidewords* to a set of *parameters* of the process. Each guideword specifies a possible deviation of the design intent for that parameter. The guidewords are their interpretation are shown in Table 8.1.

The parameters are dependent on the domain. In chemical systems, the parameters typically refer to properties of the system such as flow, pressure, temperature, and time. In software systems, the parameters typically refer to system properties such as resource/information flow, and time.

A team of experts considers each activity in the process, applying each guideword to the activity to identify possible deviations of behaviour. For each guideword/activity pair, the experts combine them and ask a "What if...?" question about the behaviour that could result.

Guideword	Meaning
NOT	The complete negation of the design intent
MORE	Quantitative increase
LESS	Quantitative decrease
AS WELL AS	Qualitative increase
PART OF	Qualitative decrease
REVERSE	Logical opposite of the design intent
OTHER THAN	Substitution of complete design intent
EARLY	Earlier than expected according to clock time
LATE	Later than expected according to clock time
BEFORE	Earlier than expected according to sequence
AFTER	Later than expected according to sequence

Table 8.1: HAZOP guidewords and their generic interpretation.

As an example of how this applies to risk management, consider Example 8.1, in which a third-party application is being used to provide some functionality to a system. The result of the third-party's activities is a system that (hopefully) produces reliable, consistent behaviour that fulfils our expectations. However, there are possible deviations from this. We use the guidewords to systematically explore what these deviations may be, and to identify their root causes. Table 8.2 shows partial results for this activities. For brevity, we consider only a few guidewords. In addition, we assume that the vendor of the software is producing a generic off-the-shelf package; that is, they have not been contracted specifically by the project to complete the software.

The result uncovers several risks, including some risks that may not be thought of using simple pondering, such as the third-party application delivering more functionality than expected, implying that a license may be more expensive, or the upside risk of the vendor delivering early.

HAZOP results tables also often include a *recommendation* for action, although this is part of risk control, which we address in Section 8.2.2. In addition, we must then use our experience to estimate the probability of the possible causes, and the impact of the consequences to the project. For example, we may estimate the probability of the vendor delivering early as being 0.000001; this is pretty accurate if my experience is anything to go by.

Risk Analysis

Risk analysis involves several sub-tasks, and which of these that a team performs is dependent on the experience of the staff, and the project itself.

The two most important parts of risk analysis are:

- **estimating the risk probability** — calculating the probability that the risk will occur;
- **estimating the risk impact** — the impact that the risk will have on the project;
- **identifying the root cause** — it is important that one identifies the *root causes* of all risks. It is common that several risks can be traced backed to a single root cause. If this root cause can be identified, then all of these risks can be controlled by addressing the single root causes.

To estimate the probability and impact, past projects and expert judgement are highly useful, but other techniques and sources of information can be used.

Identifying the root cause can be a more systematic analysis. This can be performed by taking the risky event, and working backwards to analyse the events that lead up to this.

Guideword	Interpretation	Possible Causes	Consequences
MORE	Vendor delivers additional functionality to what was expected.	Additional functionality desired by other users.	A license for the software may cost more than expected.
LESS	Some expected functionality missing.	Misunderstanding of customer base by vendor.Misunderstanding of the vendors intentions by the project team.	Package will not be sufficient for the needs of the project. Additional functionality will need to be developed in house.
OTHER THAN	Package delivers functionality completely inconsistent with expectations.	Misunderstanding of the vendors intentions by the project team.	Another package will need to be sourced or developed in house.
EARLY	Vendor delivers the package prior to expected date.	Vendor is well-organised.	Some activities can be brought forward in the schedule.
LATE	Vendor delivers the package after the expected date.	Inaccurate planning by vendor. Problems may exists with the package.	Some activities will be delayed.

Table 8.2: A partial report from a HAZOP study of Example 8.1.

Risk Prioritisation

The final step in risk assessment is to *prioritise* the risks. This is where using a quantifiable scale for risk impact is useful: it results in a total order between risks based on risk exposure, so prioritising risks is as straightforward as calculating the exposure, and ranking the higher exposure risks as higher priority.

It is clear that higher priority risks require more attention and control, and that some of the lower priority may end up requiring no attention at all. These decisions must be made as part of the project. A $10K loss for one project may end in the developers losing them jobs and the company filing for bankruptcy, while for another, a $10K loss may be hardly noticed.

Once all risks in a project have been identified, they are classified and placed in a risk register/log. Classification is important, because what we do not want is simply a list of risks in some arbitrary order. Classifying risks allows us to define responses to the risks for closely related risks. It is not uncommon that a single strategy for dealing with a risk can also mitigate several related risks.

8.2.2 Risk Control

Once we have identified our risks and exposure that our project has to them, we must determine how we are going to control them. There are four general techniques for risk reduction:

1. *Avoid* the risk. This means that we completely prevent the risky event from occurring, by either ensuring its probability is 0, or ensuring its impact 0.

2. *Mitigate* the risk. This involves employing techniques to reduce the probability of the risk, or reduce the impact of the risk. This results in a *residual risk* — that is, a risk consisting of the same event, but with a lower probability/impact, and therefore low exposure. We then must analyse the residual risk as we would our primary risk.

3. *Transfer* the risk. This involves transferring the burden of the risk to another party. Insurance is one example of risk transfer, in which the impact of the risk is offset by payments from the insurer.

4. *Accept* the risk. This means that we believe that the risk is of an acceptable exposure, that we hope that the event does not occur, or that the risk exposure is less than the cost of any techniques to avoid, mitigate, or transfer it.

Controlling risks can be done *reactively* or *proactively*. In the former case, the plan for dealing with a risk is to wait until the event occurs, and then set about implementing the strategy to control it. In the latter case, the plan for dealing with a risk is to identify the situations that may arise immediately before an event occurs, and to try to prevent the event from occurring, or reduce the impact it has on our project.

Proactively controlling risks is the preferred option, because it may help to prevent the event from occurring. However, it requires us to monitor the project more closely, so is likely to be more costly.

Example 8.4. Controlling the risk of third-party software applications

We return to our favourite risk example of the third-party software application. In this case, there are many different risks that may require different techniques to control them, however, we discuss only one example of each of the four techniques for controlling risks.

We can *avoid* all of the risks associated with the third-party application by not using the application all together. This may involve developing the required functionality in house, rather than buying it. However, this creates a whole new set of risks that must be assessed and controlled. Risk avoidance is very much proactive risk management. Alternatively, we can change the requirements of our own system, therefore removing the need to the third-party application at all. This is also a good example to contrast the difference between risk and uncertainty. The third-party application is an off-the-shelf system, so will be developed anyway. If we do not use it, we may still be uncertain about many things related to the application, but these are no longer risks to our project because the impact is zero. On the other hand, if we were considering contracting the vendor to build a custom system, by choosing not to use them, the probability of any risks related to the system become zero — we are no longer uncertain.

We can *mitigate* the risk of the third-party application being delivered late. First, we can reduce the probability of this by requesting a delivery date that is well before the required date; that is, by including some contingency in the plan. Alternatively, we can reduce the impact of the risk by designing the system such that the third-party application is accessed via a standard interface, and by producing a dummy implementation of that interface that allows development to continue if the third-party application is delivered late. Both of these are proactive risk management approaches.

If we have contracted the third-party to produce the software, we may *transfer* the risk of a late delivery by fixing the price of the software, and specifying in the contract that any costs resulting from late delivery of the system will be paid for by the vendor of the third-party application. This is a proactive technique. A reactive approach would be to transfer the risk by litigating against the vendor.

We can simply *accept* the risk because we have dealt with the vendor many times, and they in the past, they have produced reliable, correct software on time. Therefore, the probability of the risk is low. If the impact is also low, then the exposure will be low.

A single risk may be controlled using any of the four techniques above, including many instances of those four techniques (e.g. there may be several ways to mitigate a single risk), or a combination of them. For example, we may mitigate a risk by reducing its impact, but also take out insurance against it.

Decisions must be made as to the appropriate way to control a risk. Most importantly, one must consider the *cost* of controlling the risk, relative to the expected cost of the risk itself.

Definition 8.4. Risk Reduction Leverage

Risk reduction leverage is used to assess the effectiveness of risk control. Given an initial risk exposure, a technique for dealing with that risk, and a residual risk exposure (that is, the exposure to the risk after reduction), we can calculate whether this technique is cost effective using the following formula:

$$\text{Risk reduction leverage} = \frac{\text{Initial Risk exposure} - \text{residual risk exposure}}{\text{Cost of risk reduction}}$$

If the risk reduction leverage is less than 1, then the cost of implementing the risk is higher than the benefit gained from it (the difference in risk exposure), and is therefore not cost effective enough to implement. Either a cost-effective alternative will need to be found, or the risk will have to be accepted.

Risk reduction leverage also allows us to compare different plans for risk reduction. That is, the higher the risk reduction leverage is for a particular strategy of risk control, the more cost effective it is.

Example 8.5. *Regression testing*

As an example of using risk reduction leverage to make decisions, consider the case proposed by Pfleeger and Atlee [PA06] of deciding whether to perform regression testing on a system.

Consider that we are part of a development team building a system using an incremental lifecycle process model, such as the one described in Section 3.2. After each increment, the system is released, and each new release contains more functionality than the previous. The new functionality is relatively independent of the previous release.

Working under tight deadlines and a restricted budget, it becomes clear that some corners need to be cut to make the next release date. One possibility is that *regression testing* can be skipped. Regression testing is the process of running *all* system tests after a change, even if those tests are deemed to be unrelated to the part of the system containing that change. Empirical evidence suggests that changing one part of a system may results in changes in other parts of the system that the developers did not think of. The dependencies and interactions between different parts of systems are often so complex that it is difficult to determine how changing one part of a system affects the rest of the system.

The question then becomes: should we perform regression testing, or should we test only the new functionality? On the one hand, the resources and cost of performing regression testing will be wasted if no new faults are found. On the other hand, we risk the new release containing a critical fault, which would be costly.

We work out some estimates based on previous releases of the system. There are three possible outcomes, whether regression testing is performed or not:

1. A critical fault is found. This means that some further debugging and development will be required. The estimated loss is \$0.5 million.

2. There is a critical fault, but it is not found. The cost of this is significantly higher, because once this fault is found, the system will have to be debugged and some re-development required, but the new system will also need to be re-distributed. The estimated total loss is \$5 million.

3. There is no critical fault in the system at all. The loss for this is 0.

Next, we estimate the probabilities of the above. Clearly, these change depending on whether we perform regression teting or not. The probability of there being no critical fault remains fixed: 20%. Regression testing cannot influence this. The probability of there being a fault and finding it is 75% if we perform regression testing, but only 25% if we do not (that is, if we test only the new functionality). The probability of there being a critical fault and NOT finding it is 5% if we do regression testing, but 55% if we do not.

Figure 8.3 outlines an analysis of this risk, including the risk exposure for each of the outcomes, and the *combined risk exposure* for both of our options (performing regression testing or not).

The final estimate required is the cost of the regression testing itself. We estimate this to be \$1.25 million.

The risk reduction leverage can be calculated directly from this:

$$\begin{aligned}\text{Risk reduction leverage} &= \frac{\text{Initial Risk exposure} - \text{residual risk exposure}}{\text{Cost of risk reduction}} \\ &= \frac{\$2.875m - \$0.625m}{\$1.25m} \\ &= 1.8\end{aligned}$$

With the risk reduction leverage being greater than 1, we have estimated that our regression testing should be cost effective. If the cost of the regression testing was estimated to be greater than \$2.25 million (the cost of reducing the risk), this would not be cost effective, so we would accept the risk and not perform regression testing.

8.3 Crisis Management

As a final note of this chapter, we briefly discuss *crisis management*, and how this differs from risk management. First, we define crisis management.

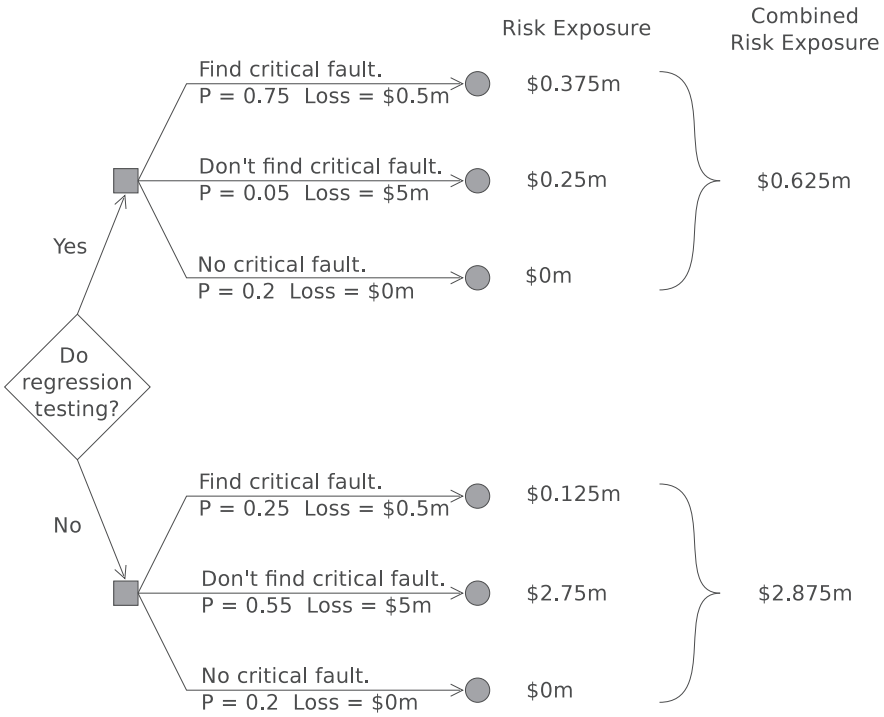

Figure 8.3: An example of calculating combined risk exposure of regression testing an application before release.

Definition 8.5. Crisis Management.
Crisis management is a process for dealing with an unpredicted event that has a negative impact.

Immediately, we can identify two differences between risk management and crisis management from this definition: 1) positive impacts are not considered; and 2) the event is *unpredicted*.

It is generally point 2 that means the difference between a risk and a crisis. A crisis erupts when an event occurs, and we have no plan for dealing with that event. As such, it can be considered that a risk becomes a crisis when the plan for dealing with that risk fails. A poor risk assessment is likely to lead to more crises than a good risk assessment.

Crisis management and risk management require different techniques and skills to successfully employ. The very fact that a crisis is unpredicted implies that we have not thought about how to deal with it, unlike a risk. A degree of crisis management is likely to be present in most projects, because even the best risk managers will overlook a critical problem on a large-scale project.

References

[CW96] C. Chapman and S. Ward. *Project risk management: processes, techniques and insights*. John Wiley, 1996.

[Kle86] T. A. Kletz. HAZOP & HAZAN notes on the identification and assessment of hazards. The Institution of Chemical Engineers, London, 1986.

[PA06] Shari L. Pfleeger and Joanne M. Atlee. *Software Engineering: Theory and Practice*. Prentice–Hall International, 3rd edition, 2006.

[Roo93] P. Rook. Risk management for software development. ESCOM Tutorial, Good synthesis of risk management approaches, 1993.